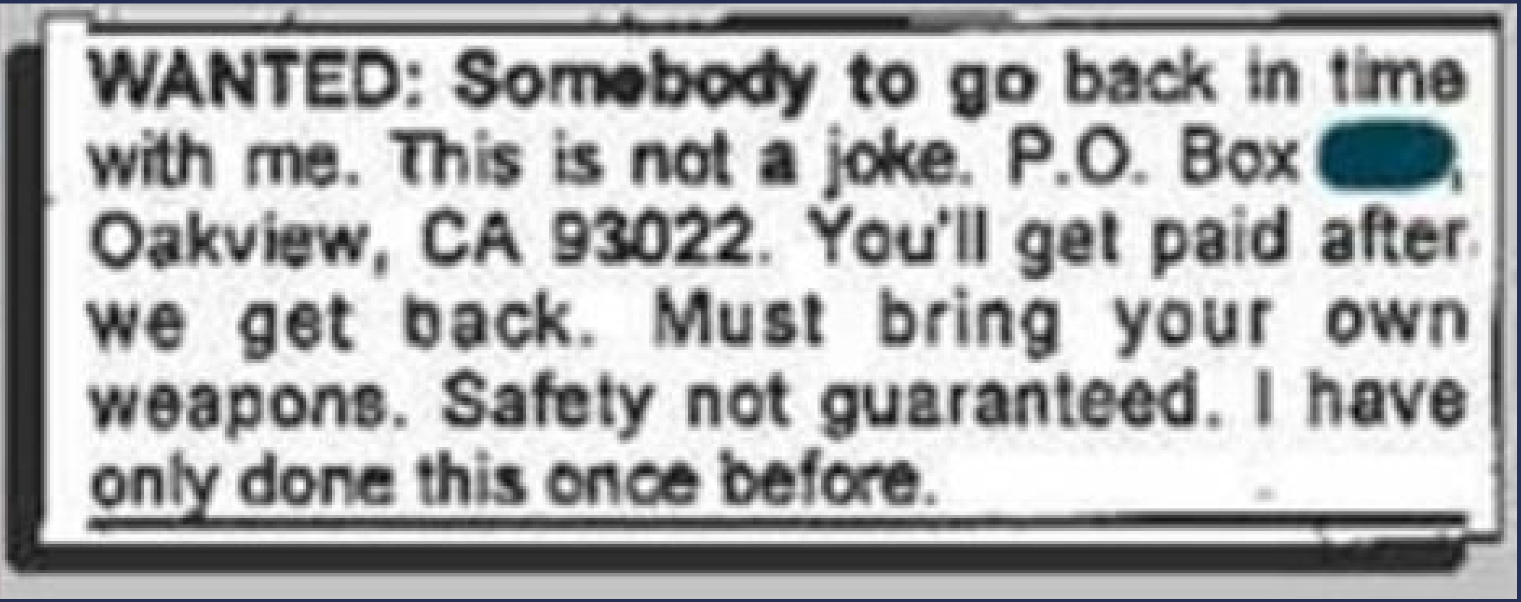
Ruby Security Training

Disclaimer



WANTED: Somebody to go back in time with me. This is not a joke. P.O. Box [redacted], Oakview, CA 93022. You'll get paid after we get back. Must bring your own weapons. Safety not guaranteed. I have only done this once before.

Outline

1. Rails Vulnerabilities
2. Root Problems
3. Possible Solutions

Rails Vulnerabilities, A Look Back

- CVE-2012-5664
- CVE-2013-0155
- CVE-2013-0156
- CVE-2013-0333

CVE-2012-5664

```
Post.find_by_id(params[:id])
params[:id] = {:select => ‘; INSERT INTO admins …’}
```

params

- ActionDispatch::Middleware::ParamsParser
- Sych. params is a HashWithIndifferentAccess.
- Every params key is coerced into a String.
- **Ruby idiom:** method options are always Symbol Hashes.

```
Post.find_by_id('select' => 'SQL ...') # NOPE
```

What about session?

User.find_by_id(session[:user_id])

- authlogic (CVE-2012-6497)
- sessions comes from the cookie
- Unfortunately, the cookie is signed with a secret token
    - config/initializers/secret_token.rb

Not Exploitable

But it got people looking

CVE-2013-0155

```
unless params[:token].nil?
  user = User.find_by_token(params[:token])
  user.reset_password!
end
```

- Truthiness and Nulliness.
- [nil], [""] and [[]] are all not nil and not empty.

CVE-2013-0156

- M-m-m-multiple vulnerabilities!
- [ActionDispatch::ParamsParser] supports parsing JSON, YAML and XML encoded request params.
- ActiveSupport::XmlMini allows for specifying types of elements
    - type="symbol" and type="yaml"

CVE-2013-0333

- Fact: JSON is a subset of YAML.
- Fact: does not mean you should parse JSON with YAML.
- Fact: Rails 2.3.x and 3.0.x did just that.

ActiveSupport::JSON::Backends::Yaml

- convert_json_to_yaml scans through the JSON, converting JSON syntax into YAML equivalents.
- Decodes the converted JSON using YAML.load.
- Does not decode escaped hex/unicode Strings **before** converting the JSON.
- Does not validate the input is well formed JSON.

YAML Exploitation

YAML Syntax

- Commonly used as a more readable version of JSON:

```
---
foo:
  - bar
  - baz
```

- But also allows serializing/deserializing arbitrary Objects:

```
---!ruby/object
  x: 1
  y: 2
```

But wait there is more!

- Override the class for String. Will call AwesomeString.new(value).

```
--- !ruby/string:AwesomeString “foo bar baz”
```

- Override the class of Hashes. Will create a new AwesomeHash object and call #[]= to populate it.

```
--- !ruby/hash:AwesomeHash
  key: “hello\nworld”
```

What To Look For

- Are there classes that extend String but treated differently. (ex: Arel::Nodes::SqlLiteral)
- Classes that define #[]= that pass the key or value to eval or system.

Protip

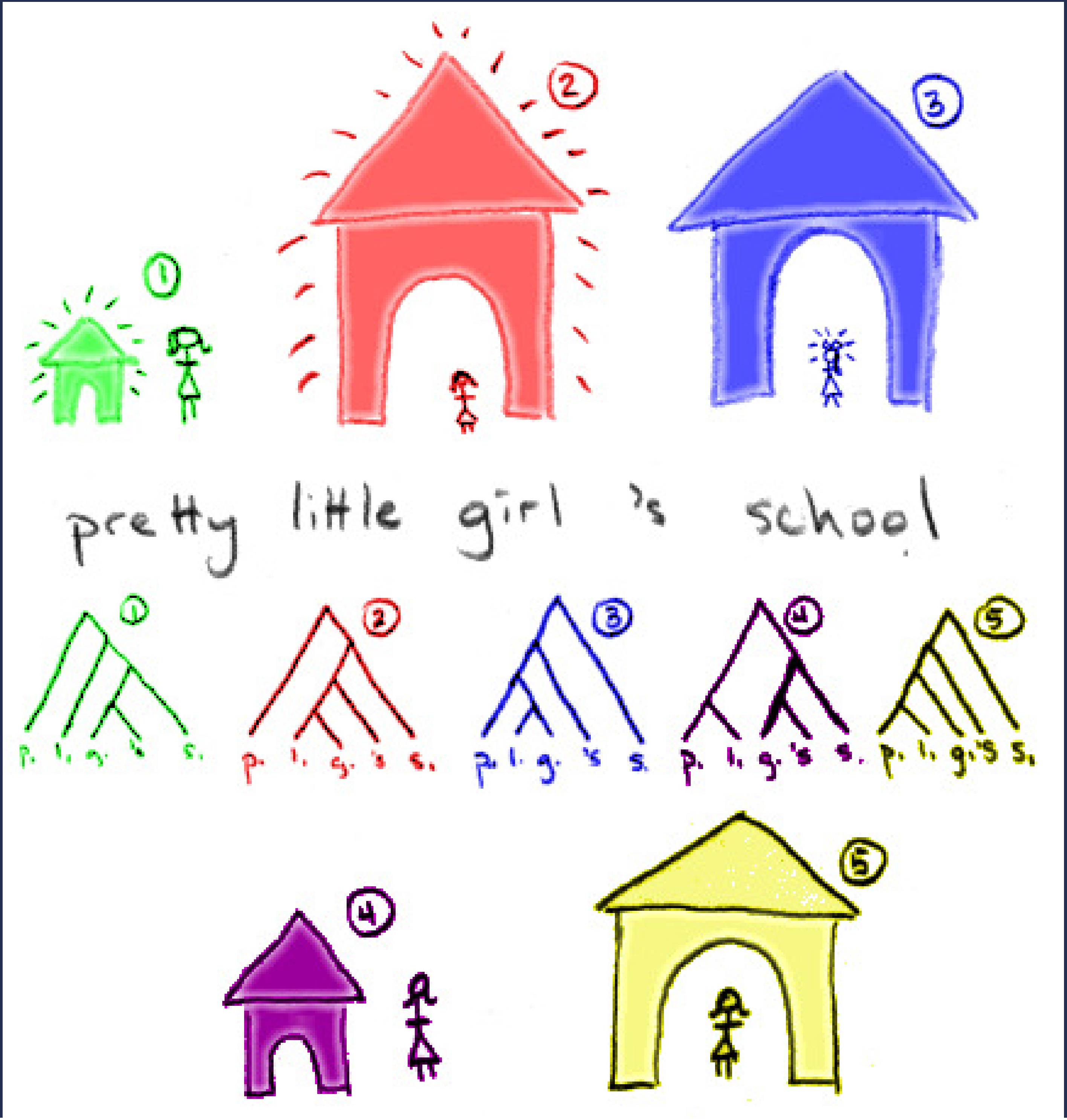
- Find all Hash-like classes:

```
ObjectSpace.each_boject(Class).select { |klass|
  klass.instance_methods.include?(:[]=)
}
```

Exercises!

- Your mission if you choose to accept it:
  - Exploit a series of trivial Sinatra web-apps that accept YAML input.
  - Each app requires an increasingly nuanced method of exploitation.
  - Update the exploit.rb file until you can successfully execute puts 'hello'.

Parsing



①
②
③
pretty little girl 's school
①
p. l. g. 's s.
②
p. l. g. 's s.
③
p. l. g. 's s.
④
p. l. g. 's s.
⑤
p. l. g.'s s.
④
⑤

Why Are We Even Covering This?

- Remember CVE-2013-0333 ?
- Using a single regular expression does not scale for parsing input.
- Simply tokenizing the input is not enough. ActiveSupport::JSON::Backends::Yaml even used the very handy StringScanner class.
- We need **full recognition**.

Parsers

- LALR (Look-Ahead Left-Right) parsers:
  - yacc / bison
  - racc
- PEG (Parsing Expression Grammar) parsers:
  - TreeTop
  - Citrus
  - Parslet

Parslet Parsers

- Define parsers by combining pattern matching rules:

```
class EmailParser < Parslet::Parser
  rule(:space) { match('\s').repeat(1) }
  rule(:space?) { space.maybe }
  rule(:dash?) { match['_-'].maybe }

  rule(:at) {
    str('@') |
    (dash? » (str('at') | str('AT')) » dash?)
  }
  rule(:dot) {
    str('.') |
    (dash? » (str('dot') | str('DOT')) » dash?)
  }

  rule(:word) { match('[a-z0-9]').repeat(1).as(:word) » space? }
  rule(:separator) { dot.as(:dot) » space? | space }
  rule(:words) { word » (separator » word).repeat }

  rule(:email) {
    (words.as(:username) » space? » at » space? » words).as(:email)
  }

  root(:email)
end
```

Parslet Parsers: Methods/Operators

- rule(:name) { ... } defines a parsing rule with the specified name.
- root :name defines which parsing rule to start at.
- str(...) matches a literal string.`
- repeat is equivalent of regex*.
- repeat(1) is equivalent to regex+.
- repeat(1,5) is equivalent to regex{1,2}.
- match(...) matches data against the specified regular expression.
- match['a-z'] is shorthand for match('[a-z']).
- | allows any one of multiple rules to be matched.
- >> requires multiple rules to be matched in succession.
- .as(:name) tags the matched text with the specified name.

Parslet Parsers: Output

```
EmailParser.new.parse(“john dot smith AT gmail dot com”)
{:email=>[
  {:username=>[
    {:word=>”john”@0},
    {:dot=>”dot”@5, :word=>”smith”@9}
  ]},
  {:word=>”gmail”@18},
]}
```

Parslet Transforms

- Transforms or Sanitize the Parslet tree into something more useful:

```
class EmailSanitizer < Parslet::Transform
  rule(:dot => simple(:dot), :word => simple(:word)) { “.#{word}” }
  rule(:word => simple(:word)) { word }
  rule(:username => sequence(:username)) { username.join + “@” }
  rule(:username => simple(:username)) { username.to_s + “@” }
  rule(:email => sequence(:email)) { email.join }
end
```

Parslet Transforms: Methods

- rule(:name => ...) { action } defines a pattern matching rule for the Parslet tree.
- Parslet automatically walks the intermediary tree for you and performs action whenever it encounters pattern
- simple(:name) matches a single String value.
- sequence(:name) matches an Array of String values.
- subtree(:name) matches a Hash within the Parslet tree.

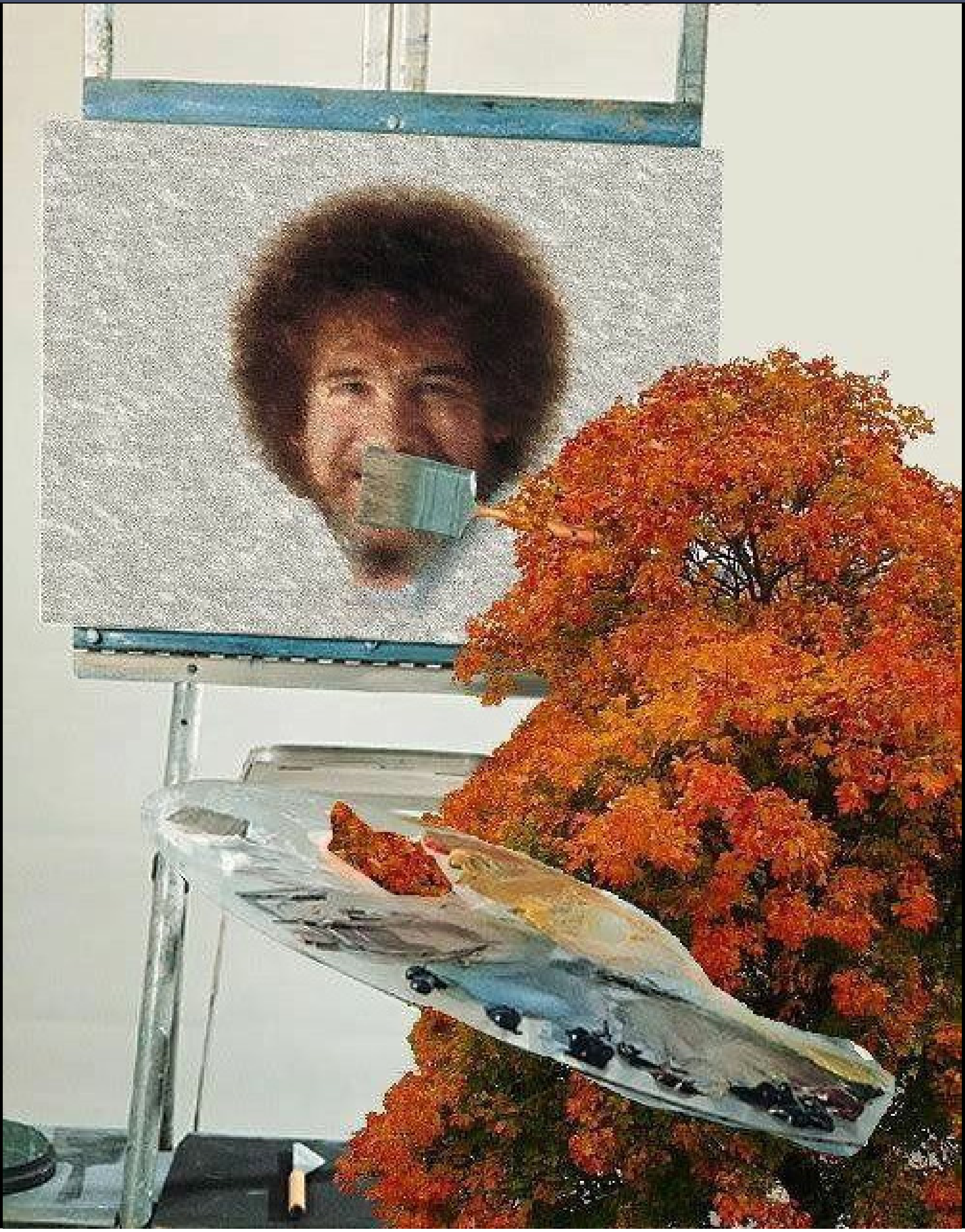
Parslet Transforms: Output

EmailSanitizer.new.apply(EmailParser.new.parse(“john dot smith AT gmail dot com”)) # => “john.smith@g

Exercises!

- Your mission if you choose to accept it:
  - level1: write a JSON parser using Parslet.
  - level2: write Parslet Transforms to remove [nil], [""] and [[]].
  - Boilerplate code and RSpec tests provided.

Mutation Testing



Problem

- How do we know when we have tested every edge-case?
    - Unit vs. Integration?
    - Test coverage metrics?
    - Do more tests really mean higher test coverage?
    - Input Fuzzing?
- How do we verify the correctness of the tests?

Mutation Testing: A New Approach

- Instead of fuzzing the input, what if we fuzzed the **code**? (inception sound effect here)
  1. Parse the code into an AST
  2. Permutate over every mutation of the AST.
  3. Eval mutated AST.
  4. Run tests against mutated code.
  5. Since the code is semantically different, the tests should fail. If the tests do not fail, than you have code which can break and the tests will not catch it.

Mutant

- The new hotness in mutation testing for Ruby 1.9.
- Loads one or more test files and selects the classes you want to mutate.

  ```
  $ mutant -r ./spec/foo_spec.rb -r --rspec-full ::Foo
  ```
- When all mutations are “killed” by the tests, you have reached full coverage.

Exercises!

- Your mission if you choose to accept it:
  - Write RSpec tests for some annoying complex authorization code for a fictional Secure Document Database.
  - Run the rake mutant task after writing tests.
  - When mutant prints all green, you are done.

Ruby Security Tools, for Ruby

QUAKER
Life
Original
MULTIGRAIN CEREAL
You Could
WIN A TRIP!
LEGOLAND

Ecosystem

- [Egor Homakav]
- Larry Cashdollar (yes, that is his actual name)
- Rails Security
    - Michael Koziarski
    - Arron “tenderlove” Patterson
- RedHat OpenShift
    - Kurt Seifried (also runs oss-sec)
    - Ramon de C Valle (aka rcvalle)
- RubySec
    - Max Veytsman
    - Phill MV
    - Myself
    - Bryan Helmkamp
    - Charlie Summervile
    - Tony Arcieri

Tools

- brakeman
- ruby-advisory-db
- bundler-audit
- gemcanary
- ronin

Shameless Plug

- Ronin is kind of big, and kind of awesome:
  - ronin-support
  - ronin
  - ronin-gen
  - ronin-asm
  - ronin-sql
  - ronin-web
  - ronin-exploits
  - ronin-scanners
  - ronin-bruteforcers
  - I think I might have a programming problem

Exercise!

- Your mission if you choose to accept it:
  - Convert one of your YAML exploits to use the convenience methods from ronin-support.
  - Marvel at how easy it was.